Performance Optimization With Velocity Engine

Session 102

Performance Optimizations on the FP and Vector Units

A. Sazegari
Vector and Numerics Group

Introduction

- Numerics libraries in Jaguar
- Benefits of vectorization
- Vector libraries in Jaguar
- Some optimization techniques using the Vector and Floating-Point Units
- Profiling tools

What You'll Learn

- Available vector libraries
- Some optimization techniques
  - Data injection
  - Vector vs. scalar algorithms
  - Be aware of the LSU bottleneck
  - Using ISA details
- Increased parallelism is key
- How to use Apple's profiling tools

What's New in Jaguar

- Numerics
  - Libm (MathLib v5)
  - FixMath
- Vector
  - Expanded vDSP
  - Standard Basic Linear Algebra Subroutines (BLAS)
- Vectorized technologies (Quartz and ColorSync)

New libm in Jaguar

- MathLib v4 shipped in 10.1.3, brings performance parity with Mac OS 9's MathLib v3
- Coming in Jaguar: MathLib v5
- v5 Faster than v3 (Mac OS 9) and faster than v4 (10.1.3)
- v5 is IEEE-754 and C99 compliant on double precision
- Implementations
  - PPC
  - IA-32 for Darwin users

New FixMath in Jaguar

- Performance parity with Mac OS 9
- FixMath is floating-point based
- New code should not use FixMath
- Use floating-point instead

Vectorization: General Guidelines

- Profile, profile, profile, and then profile
- Use CHUD to profile
- Rewrite the scalar using data parallelism
- First rewrite select parts in vector
- Use vector libraries as much as possible
- Be aware of register spillage

An Example: IMDCT For MP3

(Inverse Modified Discrete Cosine Transform)

	Units of Time (Smaller numbers are better)
IMDCT code: (has libm Cos in the loop)	583
Original ISO: (2048 Cos table look up)	20.1
Lee Optimization: (384 Cos table)	3.45
FFT Based IMDCT: (50 Cos table)	1.23
Vector FFT Based IMDCT:	0.48
Overall improvement:	Over 1214x
Improvement over ISO:	Over 41x

Vectorization Benefits

- PowerPC vector unit has an orthogonal design
- No overhead for using the vector engine
- Pipelined single cycle instruction execution
- Library support built into the OS
  - vDSP
  - BLAS
  - vBasicOps
  - vMathLib

Demo

Vector Quartz

Ralph Brunner
Core Graphics

Demo

Vector ColorSync

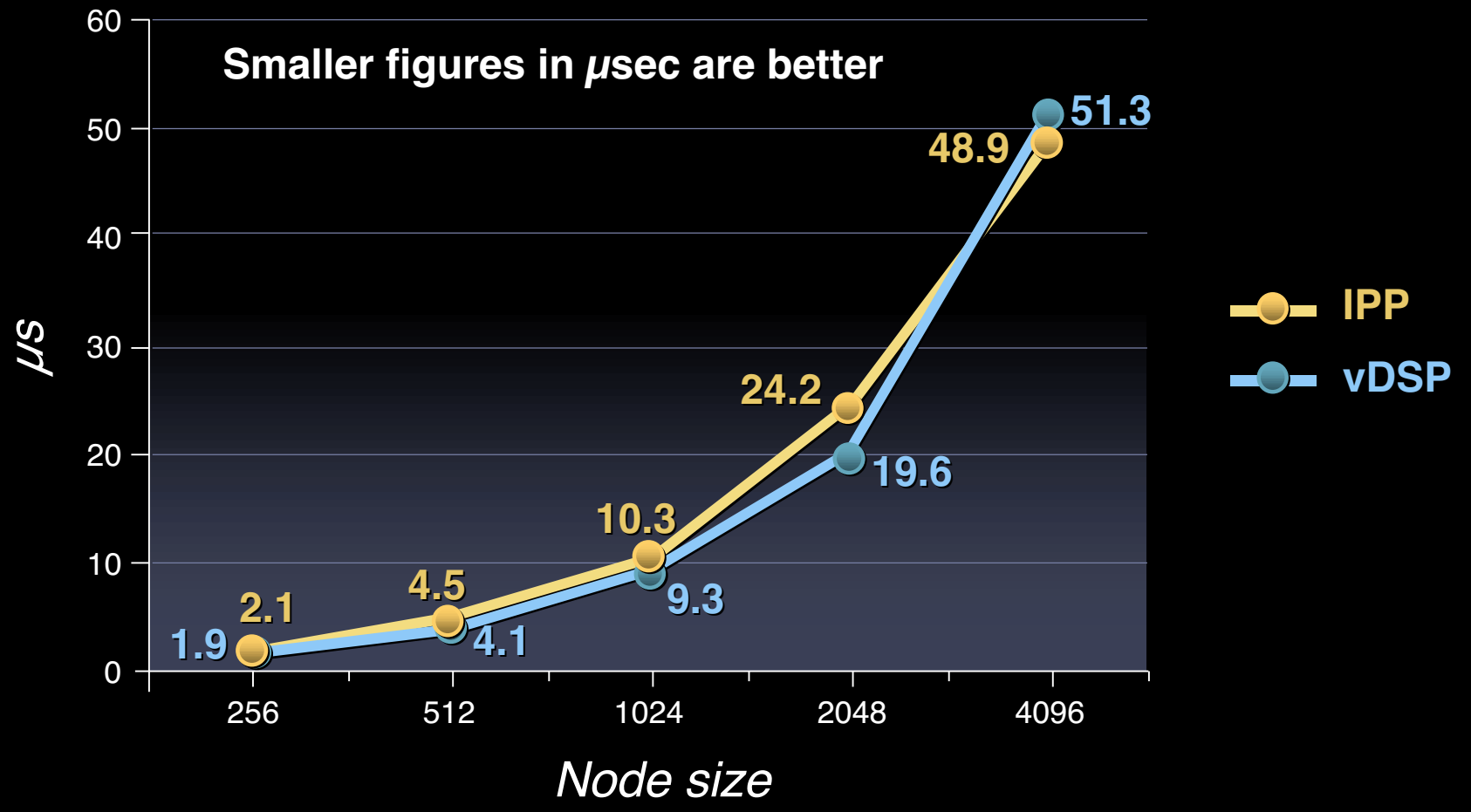
Robert Murley
Vector and Numerics Group

vDSP

- New double precision DSP functions in Jaguar
- Very high performance single and double
- Excellent single complex, real, 1D and 2D FFT
  - 1024 complex FFT takes 9.8 μs on a 1Ghz PPC
  - 1024 real FFT takes 5.2 μs on a 1Ghz PPC
- Base 3 and 5 FFTs

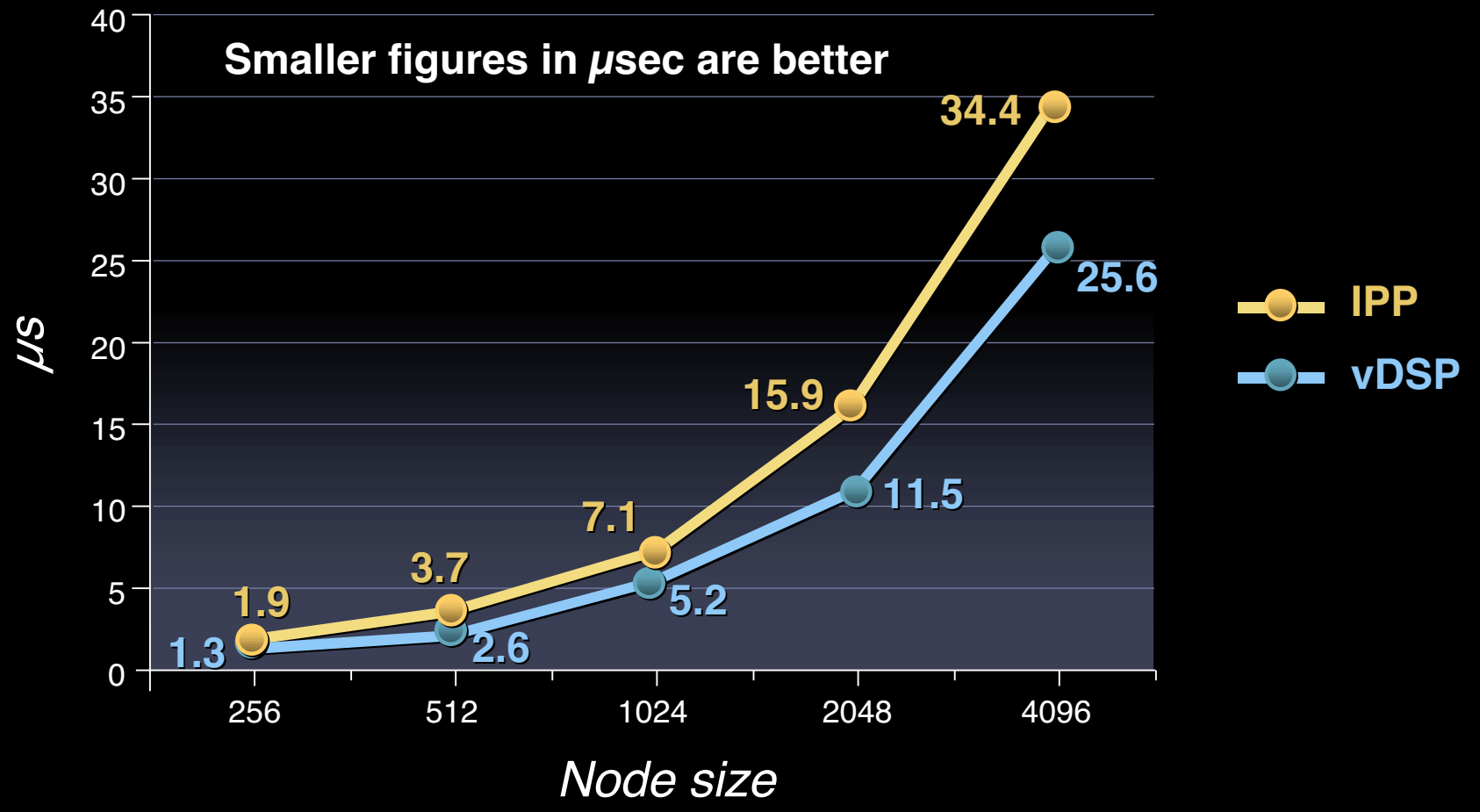
Performance of Complex 1D-FFT

Mac OS X vDSP on 1GHz vs LINUX IPP on 2.2 GHZ P4



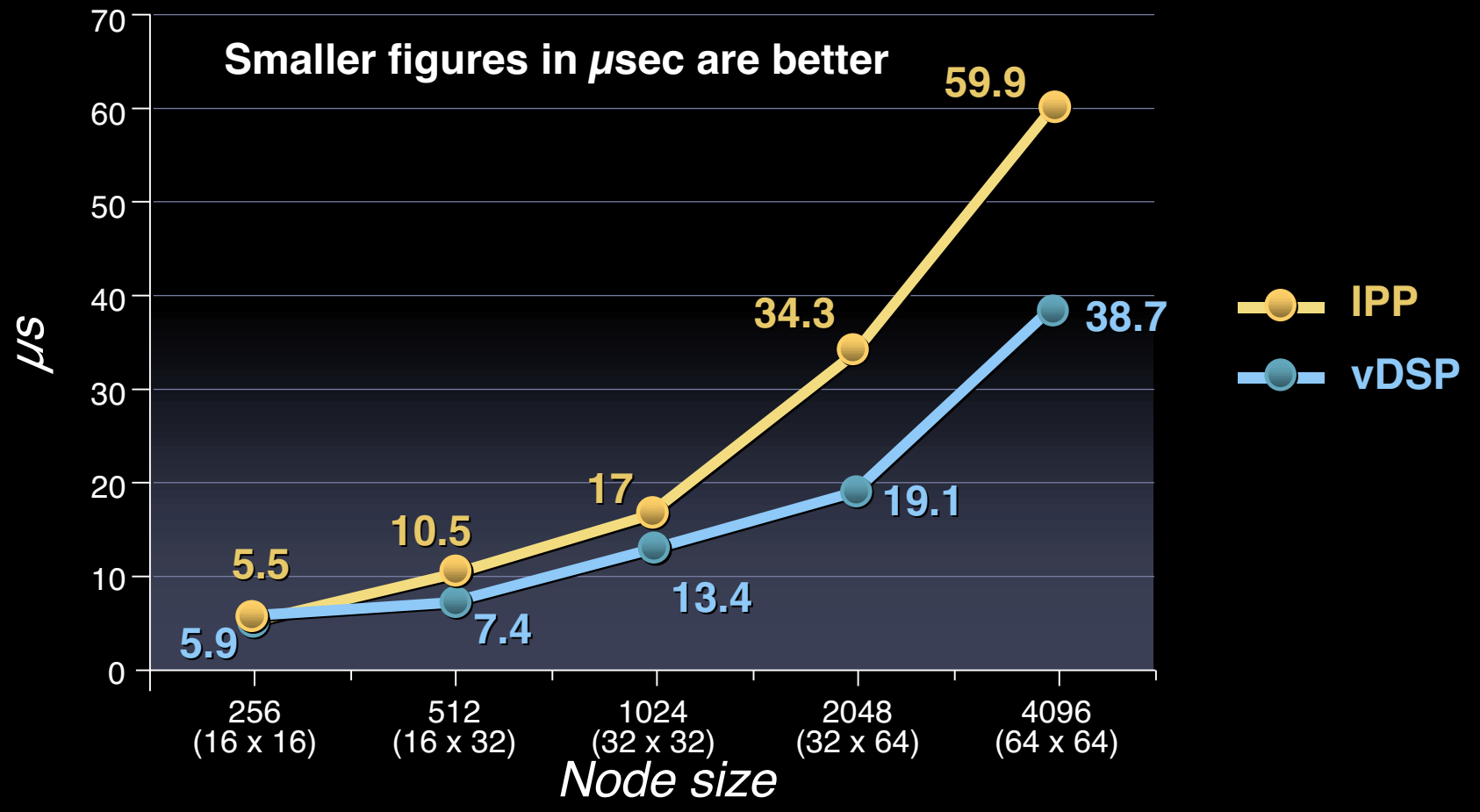
Performance of Real 1D-FFT

Mac OS X vDSP on 1GHz vs LINUX IPP on 2.2 GHZ P4



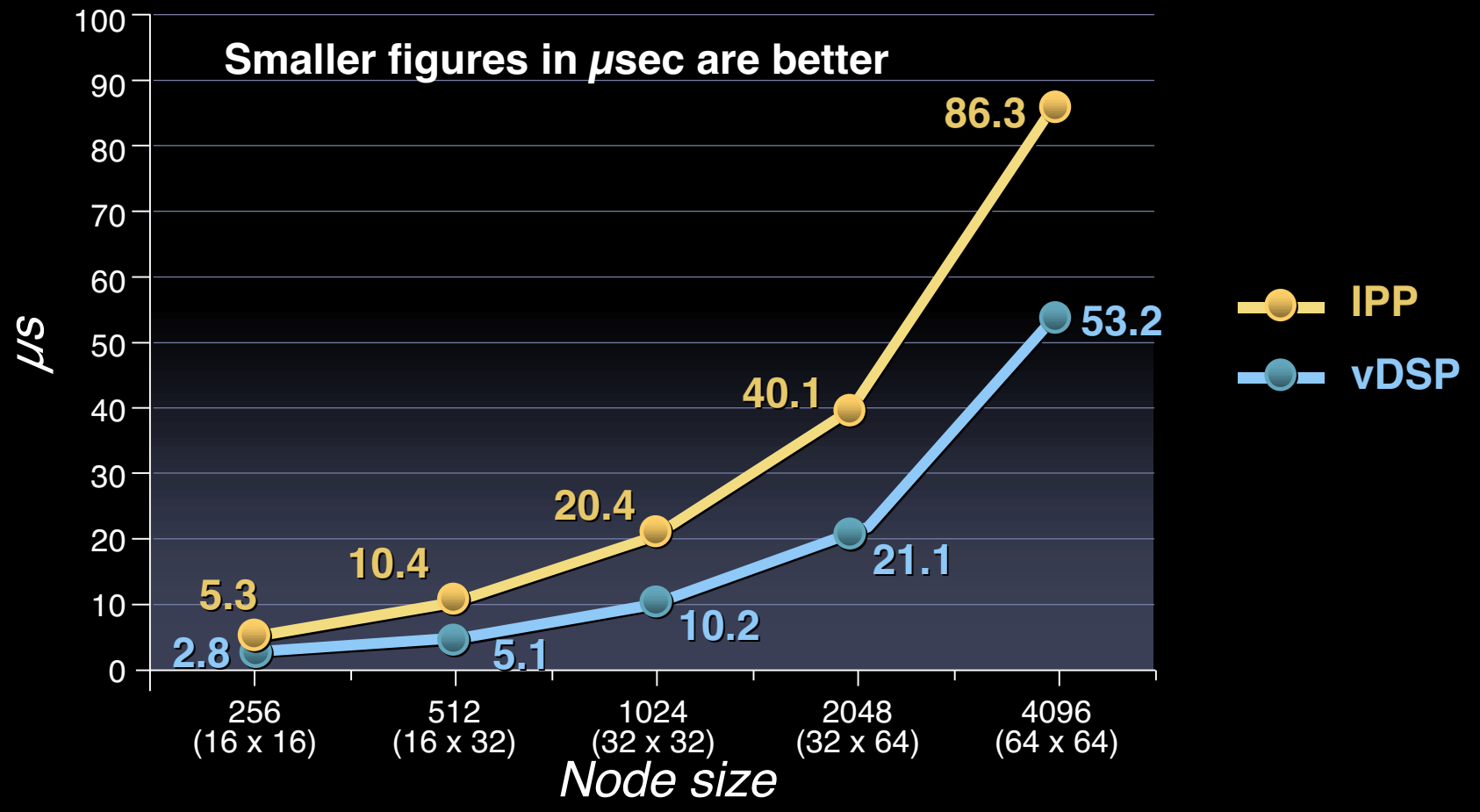
Performance of Real 2D-FFT

Mac OS X vDSP on 1GHz vs LINUX IPP on 2.2 GHZ P4



Performance of Complex 2D-FFT

Mac OS X vDSP on 1GHz vs LINUX IPP on 2.2 GHZ P4



Demo

Vector iTunes

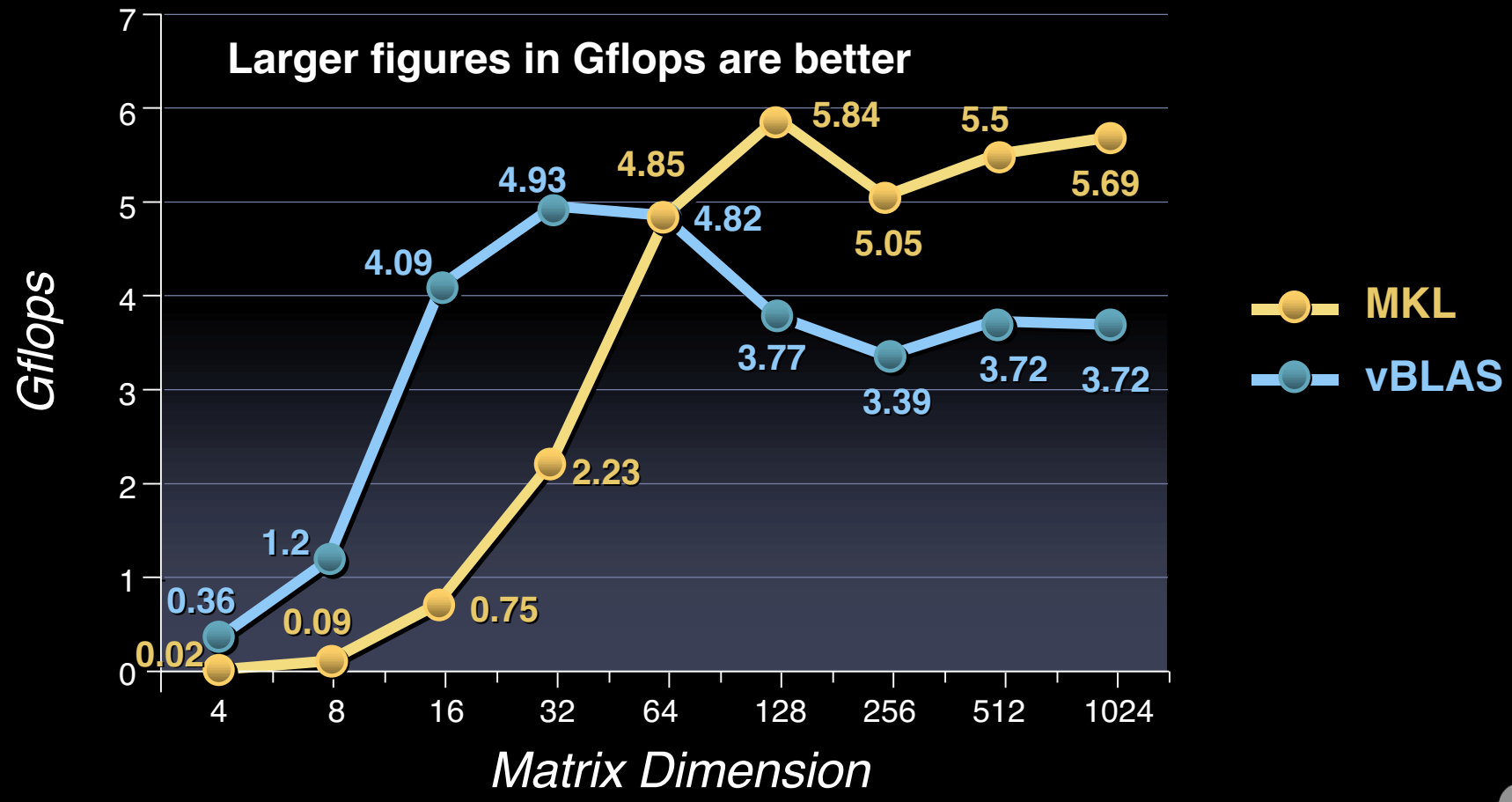
Bill Kincaid
iTunes Group

Basic Linear Algebra Subroutines (BLAS)

- Jaguar adds a full suite of BLAS
- Single and double precision
- Real and complex
- Vector and scalar
- Transparently selects between scalar or vector
- Multi-gflop float performance from DRAM
- Very fast special cases such as 4x4 matrix multiply

Performance of SGEMM()

Mac OS X vBLAS on 1GHz G4 vs LINUX MKL on 2.2 GHZ P4



BLAS vs. VectorOps Usage

- BLAS is an industry standard
- BLAS is very high performance
- vectorOps more of an educational tool
- vectorOps source available
- vectorOps is good to use with small arrays
- BLAS is preferred over vectorOps

Performance Issues: Data Injection Techniques

- A software approach to minimize pipeline stalls
  - Take the execution pipeline into account
- The deeper the pipeline, the more independent sets of data is required for efficient processing
- Dynamic dependency slows the machine down

Performance Issues: Vector Algorithms

- Some fast scalar algorithms are not vectorizable
- Different algorithms maybe required
- Dynamic dependency of instructions causes slowdowns
- Data injection techniques will not work in presence of dynamic dependencies

Performance Issues: LSU Bottleneck

- 2 or 3 load/stores for FPU/Vector instructions
- Inline or remove trivial functions
- Minimize memory movement
- Keep values in registers
- LSU over-reliance frequently causes 2–3x slowdowns

Performance Issues: ISA Details

- Be aware of the PPC Instruction Set Architecture
- Not all instructions are created equally
  - FP divide is not pipelined, flushes the pipe
  - Reciprocal estimate is not pipelined
- Just dual issue on older G4 machines
  - Can only issue to 1 subunit of VALU + VPERM
- Triple issue on newer ones

A floating-point routine operating on one set of data: {dividend1, divisor1}

```
void fdivs1 ( float *dividend1, float *divisor1 )
    {
    register float estimate1, y1, q, r;
    estimate1 = __frsqrtes ( *divisor1 );
    estimate1 *= estimate1;
    y1 = estimate1 + estimate1 * ( 1.0f - *divisor1 * estimate1 );
    q = *dividend1 * y1;
    r = *dividend1 - q * *divisor1;
    *dividend1 = q + y1 * r;
    }
```

```
0:frsqrte F10,F9          | 1 | IDEEFR..................... | 6
1:fmuls   F7,F10,F10      | 1 | IIDDDEEFR.................. | 9
2:addis   R5,R31,0x0      | 1 | IIDFFFFFR.................. | 9
3:lfs     F6,0x454(R5)    | 1 | IIIDEFFFFR................. | 10
4:fsubs   F8,F9,F7        | 2 | .IIIIDDDEEFR............... | 12
5:fmadds  F2,F7,F8,F7     | 2 | .IIIIIIIDDDEEFR............ | 15
6:fadds   F4,F2,F2        | 3 | ..IIIIIIIIDDDEEFR.......... | 18
7:fsubs   F5,F6,F4        | 4 | ...IIIIIIIIIIDDDEEFR...... | 21
8:fmadds  F1,F2,F5,F4     | 5 | ....IIIIIIIIIIIIDDDEEFR... | 24
9:mfspr   R3,PMC1         | 5 | ....IIIIIIIIIIIIDEEEEEER.. | 25
```

I = Fetch, D = Dispatch, E = Execute, F = Finished, R = Retired

This program takes 25 cycles to execute, suffers from many stalls

Same program operating on three sets of data:{dividend1,2,3, divisor1,2,3}

```
estimate1 = __frsqrtes ( divisor1 );
estimate1 *= estimate1;
estimate2 = __frsqrtes ( divisor2 );
estimate2 *= estimate2;
estimate3 = __frsqrtes ( divisor3 );
estimate3 *= estimate3;
y1 = estimate1 + estimate1 * ( 1.0f - divisor1 * estimate1 );
y2 = estimate2 + estimate2 * ( 1.0f - divisor2 * estimate2 );
y3 = estimate3 + estimate3 * ( 1.0f - divisor3 * estimate3 );
q1 = dividend1 * y1;
q2 = dividend2 * y2;
q3 = dividend3 * y3;
r1 = dividend1 - q1 * divisor1;
r2 = dividend2 - q2 * divisor2;
r3 = dividend3 - q3 * divisor3;
dividend1 = q1 + y1 * r1;
dividend2 = q2 + y2 * r2;
dividend3 = q3 + y3 * r3;
```

```
 0:frsqrte F13,F8           |  1 | IDEEFR.......................... | 6
 1:frsqrte F10,F5           |  1 | IIDEEFR......................... | 7
 2:frsqrte F9,F6            |  1 | IIIDEEFR........................ | 8
 3:fmuls   F0,F13,F13       |  2 | .IIIIDEEFR...................... | 10
 4:fmuls   F7,F9,F9         |  2 | .IIIIIDEEFR..................... | 11
 5:fmuls   F13,F10,F10      |  3 | ..IIIIIDEEFR.................... | 12
 6:fsubs   F11,F8,F0        |  3 | ..IIIIIIIDEEFR.................. | 14
 7:fnmsubs F9,F7,F6,F8      |  4 | ...IIIIIIIDEEFR................. | 15
 8:fnmsubs F10,F13,F5,F8    |  5 | ....IIIIIIIDEEFR................ | 16
 9:fmadds  F0,F0,F11,F0     |  7 | ......IIIIIIIDEEFR.............. | 18
10:fmadds  F12,F7,F9,F7     |  8 | .......IIIIIIIDEEFR............. | 19
11:fmadds  F13,F13,F10,F13  |  9 | ........IIIIIIIDEEFR............ | 20
12:fadds   F10,F0,F0        | 11 | ..........IIIIIIIDEEFR.......... | 22
13:fmuls   F7,F12,F30       | 12 | ...........IIIIIIIDEEFR......... | 23
14:fmuls   F9,F13,F31       | 13 | ............IIIIIIIDEEFR........ | 24
15:fsubs   F11,F29,F10      | 15 | ..............IIIIIIIDEEFR...... | 26
16:fnmsubs F5,F9,F5,F31     | 16 | ...............IIIIIIIDEEFR..... | 27
17:fnmsubs F31,F7,F6,F30    | 17 | ................IIIIIIIDEEFR.... | 28
18:fmadds  F29,F0,F11,F10   | 19 | ..................IIIIIIIDEEFR.. | 30
19:fmadds  F10,F13,F5,F9    | 20 | ...................IIIIIIIDEEFR. | 31
20:fmadds  F9,F12,F31,F7    | 21 | ....................IIIIIIIDEEFR | 32
```

This program takes 32 cycles to execute 3 sets of operands. There are no stalls.

Where to Go From

- Browse it
  - <developer.apple.com/hardware/ve>
- Code it
  - /System/Library/Frameworks/vecLib.framework/
- Include it
  - #include <vecLib/vecLib.h>
- Do it
  - cc -faltivec -framework vecLib file.c

Computer Hardware Understanding Development Tools

"CHUD" Tools

Nathan Slingerland and Sanjay Patel
Architecture Performance Group, Apple Computer

Instruction Tracing and Analysis

- Amber
  - Command-line instruction tracer
  - Start/stop tracing by hot-key, symbols, or instrumented code
- SimG4
  - Cycle accurate simulator of PPC7400
  - Pipeline views of executing instructions

FP Division Example—Base Case

```
void fdivs1 ( float *dividend1, float *divisor1 )
    {
    register float estimate1, y1, q, r;

    // START TRACING HERE
    StartStop_Amber();

    estimate1 = __fress ( *divisor1 ); // USE A RECIPROCAL ESTIMATE
    y1 = estimate1 + estimate1 * ( 1.0f - *divisor1 * estimate1 );
    q = *dividend1 * y1;
    r = *dividend1 - q * *divisor1;
    *dividend1 = q + y1 * r;

    // STOP TRACING HERE
    StartStop_Amber();
    }
```

SimG4 Output—Base Case

```
0:fres    F7,F8         |  1 | IDEEEEEEEEEEEEFR.................. | 16
1:fsubs   F0,F8,F7      |  1 | IIIIIIIIIIIIDDEEFR................ | 19
2:addis   R5,R31,0x0    |  1 | IIIIIIIIIIIIDFFFFFR............... | 19
3:lfs     F6,0x450(R5)  |  2 | .IIIIIIIIIIIIDEFFFFR.............. | 20
4:fmadds  F2,F7,F0,F7   |  2 | .IIIIIIIIIIIIIIDDEEFR............. | 22
5:fadds   F4,F2,F2      |  2 | .IIIIIIIIIIIIIIIIDDEEFR........... | 25
6:fsubs   F5,F6,F4      |  3 | ..IIIIIIIIIIIIIIIIIDDEEFR......... | 28
7:fmadds  F1,F2,F5,F4   | 14 | .............IIIIIIIIIIDDEEFR... | 31
```

I = Fetch, D = Dispatch, E = Execute, F = Finished, R = Retired

This program takes 31 cycles to Execute—long, non-pipelined instruction and dependency stalls.

FP Division—Using Sqrt Estimate

```
void fdivs1 ( float *dividend1, float *divisor1 )
    {
    register float estimate1, y1, q, r;

    // START TRACING HERE
    Start_StopAmber();

    estimate1 = __frsqrte ( *divisor1 ); // USE A RECIP SQ ROOT ESTIMATE
    estimate1 *= estimate1;              // AND SQUARE IT
    y1 = estimate1 + estimate1 * ( 1.0f - *divisor1 * estimate1 );
    q = *dividend1 * y1;
    r = *dividend1 - q * *divisor1;
    *dividend1 = q + y1 * r;

    // STOP TRACING HERE
    Start_StopAmber();
    }
```

SimG4 Output—Sqrt Estimate

```
0:frsqrte F10,F9          | 1 | IDEEFR..................... | 6
1:fmuls   F7,F10,F10      | 1 | IIDDEEFR................... | 9
2:addis   R5,R31,0x0      | 1 | IIDFFFFFR.................. | 9
3:lfs     F6,0x454(R5)    | 1 | IIIDEFFFFR................. | 10
4:fsubs   F8,F9,F7        | 2 | .IIIIDDEEFR................ | 12
5:fmadds  F2,F7,F8,F7     | 2 | .IIIIIIIDDEEFR............. | 15
6:fadds   F4,F2,F2        | 3 | ..IIIIIIIIDDEEFR........... | 18
7:fsubs   F5,F6,F4        | 4 | ...IIIIIIIIIDDEEFR......... | 21
8:fmadds  F1,F2,F5,F4     | 5 | ....IIIIIIIIIIIDDEEFR...... | 24
```

I = Fetch, D = Dispatch, E = Execute, F = Finished, R = Retired

This program takes 24 cycles to execute—no more fres stall, but several dependency stalls.

FP Division—Data Injection

```
void fdivs1m ( float *dividend1, float *divisor1,
               float *dividend2, float *divisor2,
               float *dividend3, float *divisor3 )
{
register float estimate1, y1, q1, r1,
estimate2, y2, q2, r2, estimate3, y3, q3, r3;

estimate1 = __frsqrtes ( divisor1 );
estimate1 *= estimate1;
estimate2 = __frsqrtes ( divisor2 );
estimate2 *= estimate2;
estimate3 = __frsqrtes ( divisor3 );
estimate3 *= estimate3;
y1 = estimate1 + estimate1 * ( 1.0f - divisor1 * estimate1 );
y2 = estimate2 + estimate2 * ( 1.0f - divisor2 * estimate2 );
y3 = estimate3 + estimate3 * ( 1.0f - divisor3 * estimate3 );
q1 = dividend1 * y1;
q2 = dividend2 * y2;
q3 = dividend3 * y3;
r1 = dividend1 - q1 * divisor1;
r2 = dividend2 - q2 * divisor2;
r3 = dividend3 - q3 * divisor3;
dividend1 = q1 + y1 * r1;           // FIRST OUTPUT
dividend2 = q2 + y2 * r2;           // SECOND OUTPUT
dividend3 = q3 + y3 * r3;           // THIRD OUTPUT
}
```

SimG4 Output—Data Injection

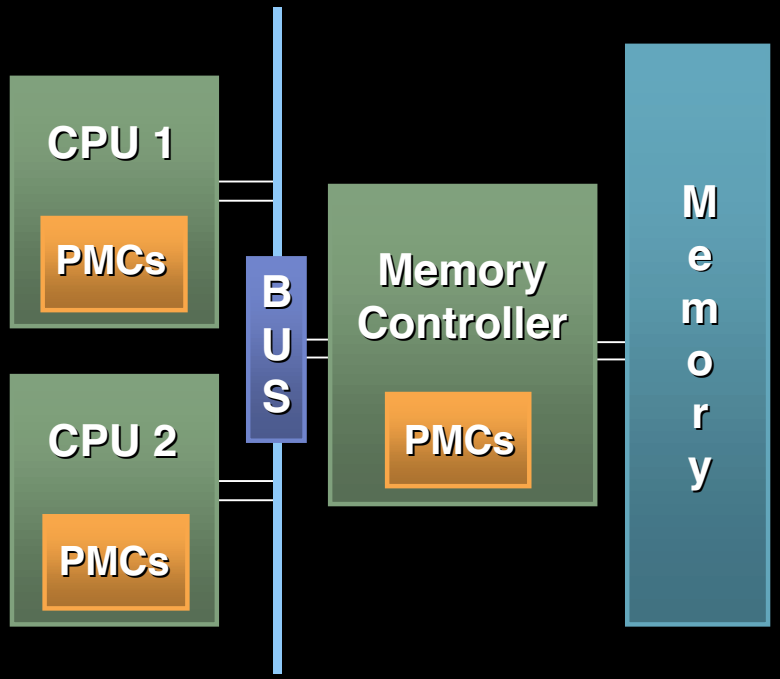
```
 0:frsqrte F13,F8           |  1 | IDEEFR.............................. | 6
 1:frsqrte F10,F5           |  1 | IIDEEFR............................. | 7
 2:frsqrte F9,F6            |  1 | IIIDEEFR............................ | 8
 3:fmuls   F0,F13,F13       |  2 | .IIIIDEEFR.......................... | 10
 4:fmuls   F7,F9,F9         |  2 | .IIIIIDEEFR......................... | 11
 5:fmuls   F13,F10,F10      |  3 | ..IIIIIDEEFR........................ | 12
 6:fsubs   F11,F8,F0        |  3 | ..IIIIIIIDEEFR...................... | 14
 7:fnmsubs F9,F7,F6,F8      |  4 | ...IIIIIIIDEEFR..................... | 15
 8:fnmsubs F10,F13,F5,F8    |  5 | ....IIIIIIIDEEFR.................... | 16
 9:fmadds  F0,F0,F11,F0     |  7 | ......IIIIIIIDEEFR.................. | 18
10:fmadds  F12,F7,F9,F7     |  8 | .......IIIIIIIDEEFR................. | 19
11:fmadds  F13,F13,F10,F13  |  9 | ........IIIIIIIDEEFR................ | 20
12:fadds   F10,F0,F0        | 11 | ..........IIIIIIIDEEFR.............. | 22
13:fmuls   F7,F12,F30       | 12 | ...........IIIIIIIDEEFR............. | 23
14:fmuls   F9,F13,F31       | 13 | ............IIIIIIIDEEFR............ | 24
15:fsubs   F11,F29,F10      | 15 | ..............IIIIIIIDEEFR......... | 26
16:fnmsubs F5,F9,F5,F31     | 16 | ...............IIIIIIIDEEFR........ | 27
17:fnmsubs F31,F7,F6,F30    | 17 | .................IIIIIIIDEEFR...... | 28
18:fmadds  F29,F0,F11,F10   | 19 | ...................IIIIIIIDEEFR.... | 30
19:fmadds  F10,F13,F5,F9    | 20 | ....................IIIIIIIDEEFR... | 31
20:fmadds  F9,F12,F31,F7    | 21 | .....................IIIIIIIDEEFR.. | 32
```

This program takes 32 cycles to execute 3 sets of operands. There are no stalls.

Summary

- You can use SimG4 to identify performance bottlenecks in the processor
- Using reciprocal square root improved performance 29%
- Using data injection gave us a 2.9x throughput improvement over the base case

Performance Counters



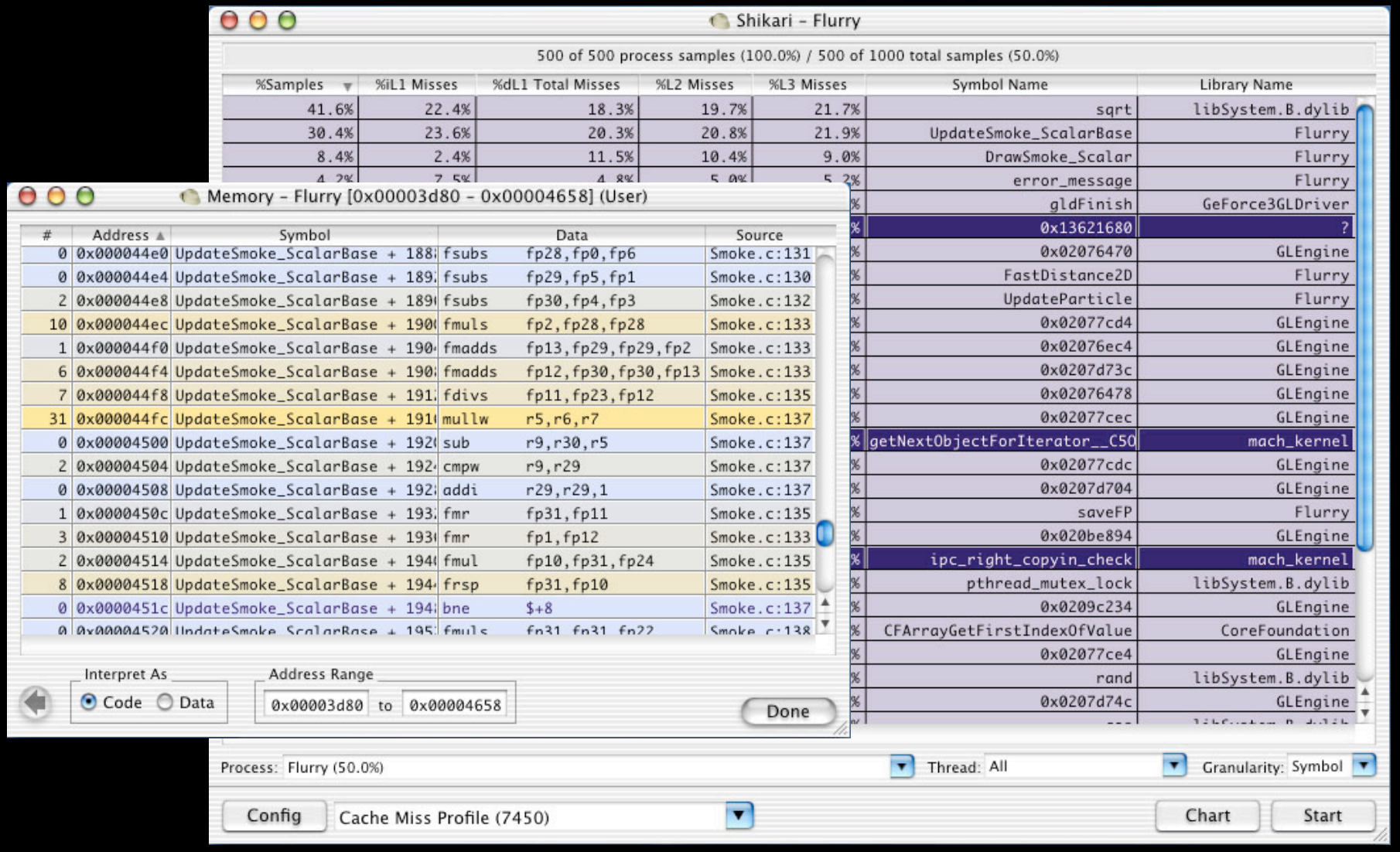
- Macintosh hardware has built-in Performance Monitor Counters (PMCs)
- Count "performance events"
  - Instructions
  - Cache misses
  - Execution stalls
- Use CHUD tools to ask the PMCs what's going on

Shikari

shi·ka·ri (sh-kär, -kr) *n. pl.* **shi·ka·ris,** a big game hunter; a professional hunter or guide

- Sample what's running on the system
  - Use event counts or timed intervals
  - System wide, process, or thread scope
- Relate performance events to your source code
  - Performance "hot spots"
  - Annotated disassembly
- Performance event histograms

Shikari Demo



Other CHUD Tools

- MONster—collect and view PMC data
- Acid—filters traces for perf statistics
- Reggie—examine and modify SPRs
- CHUD.framework—make custom perf. tools

Where to Get CHUD Tools

- Available on the web:
  http://developer.apple.com/tools/debuggers.html
  (then click on **CHUD Tools** to download)
- Report any issues to:
  chud-tools-feedback@group.apple.com

Roadmap

905 Apple Performance Tools	Hall 2 **Thurs., 5:00pm**
906 Developing for Performance	Hall 2 **Fri., 9:00am**
516 Graphics and Imaging Performance Tuning	Hall 2 **Fri., 3:30pm**
112 Writing Threaded Apps on Mac OS X	Room J **Thurs., 9:00am**

Who To Contact

Mark Tozer-Vilchez
Hardware Evangelist
tozer@apple.com

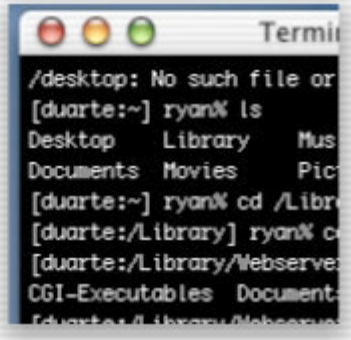
http://developer.apple.com/wwdc2002/urls.html

Technical Documentation

http://developer.apple.com/hardware/ve/index.html

http://developer.apple.com/hardware/ve/acgresearch.html

Q&A



Mark Tozer-Vilchez
Hardware Evangelist
tozer@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002